Each guideline describes a good practice, and intention is to have a overall consistent presentation. **These guidelines are derived from angular style guide. Intention here is to have a brief guide which is easier for developer to refer & follow.** you can find more details on why we need to follow these guidelines & for complete guide, please refer angular style guide here.

Strategies

Single responsibility : Apply the *single responsibility principle* (SRP) to all components, services, and other symbols. This helps make the app cleaner, easier to read and maintain, and more testable.

- **Do** one thing
- **Do** define small functions
- **Do** create services with a single responsibility that is encapsulated by its context.
- **Do** limit logic in a component to only that required for the view. All other logic should be delegated to services.
- **Do** move reusable logic to services and keep components simple and focused on their intended purpose.
- **Do** put presentation logic in the component class, and not in the template.

Folder Structure

There are many ways to define folder structure, but to keep navigation easy and scalable we need to follow basic guidelines.
Modules
First key decision to make is to list down the modules by splitting them based on the features, resuability, standalone feature, dependency mapping
- **Core Module** - takes the role of root module which is inherited by your application modules & components. Your layout/shell components (header, footer, nav), authentication, logger, localization, http interceptors should go here.
- **Shared Module** - All your resuable components, directives, pipes, services, interfaces.will go here. This means Angular CommonModule will also go here as it will used by all other components. Common components like pipesm, Date Picker Component
Loader Component,Pagination, Router Transition animations should go here.
- **Application Module** - must be split based on the business functionality, res-usability, standalone feature set, minimal or no dependencies between features, plug & play within or across applications. **Do** create folders named for the feature area they represent.

File structure conventions

Some code examples display a file that has one or more similarly named companion files. For example, `hero.component.ts` and `hero.component.html`.

The guideline uses the shortcut `hero.component.ts|html|css|spec` to represent those various files. Using this shortcut makes this guide's file structures easier to read and more terse.

Naming Conventions

Naming conventions play an important role in application consistency & readability. In general **Do** use consistent names for all assets named after what they do/represent/meant. To keep the application consistent please follow the below guidelines.

- **Do** use consistent names for all symbols. Do follow a pattern that describes the symbol's feature then its type. The recommended pattern is `feature.type.ts`
- **Do** use dots & dashes to separate words in the descriptive name.
- **Do** use *dashed-case* or *kebab-case* for naming the element selectors of components
  example: `app/heroes/hero-list.component.ts`
- **Do** use upper camel case for ClassNames, ModuleNames, Interfaces, Pipes & Services
  Example : class `AppComponent, AppModule, HeroCollectorService`
- **Do*** use lower camel case for variable/Property/method names.
  Example : const `windowWidth`
- **Do** name test specification files the same as the component they test. suffix unit testing files with `.spec` & end to end testing files with `.e2e-spec`.
  Example : `heroes.e2e-spec.ts` `heroes.service.spec.ts`
- **Do** make sure, all Modules have suffix Module
  Example: `AppModule`

Coding

- **Do** extract templates and styles into a separate file, when more than 3 lines.
- **Consider** leaving one empty line between third party imports and application imports
- **Do** use the @Input() and @Output() class decorators instead of the `inputs` and `outputs` properties of the @Directive and @Component metadata

- **Do** place properties up top followed by methods.
- **Do** place private members after public members, alphabetized.

Comments

Comments are important for the long term span of the application. So please make sure to add the comments appropriately in each file.
Few scenarios where comments are must:

- In each file about what it does in the beginning of the file.
- Hide and show scenario where DOM is getting manipulated.
- Where the interaction between the components happen
- During Sync/Async/ service calls & callback invoke.

Linting

Linting is the basic coding standard validation expected these days. Please make sure your application doesn't have any linting errors.
use the following command to make sure the application is lint error free

```
ng lint
```

If there are any errors please fix them during the development itself. You can use the following command to fix some of the linting errors.

```
ng lint --fix
```

AOT(Ahead of Time)

AOT compilation is must for the production build. Make sure it is working all the time.

```
ng buid --prod
```

Make sure to check AOT build during the following scenarios

- When a **third party library** is integrated/used. If the library doesn't support AOT, its going to cause the issue later, so its better to check during the integration itself.
- During **code integration** and clean up
- Any **package.json** & **webpack** change
- Check before **checking in**

Services

Services are backbones for the application where application's business logic should reside. Please follow below guidelines while creating a service.

- Use **angular CLI** to create a service, @Injectable metadata in it allows Angular to optimize an app by removing the service if it turns out not to be used after all.
- For every **subscribe** call, make sure **unsubscribe** is called, unless you are using async pipe which does it by default.

Unit Testing

Unit testing is integral part of development these days. So make sure you write unit test cases with your development. Even if unit testing is **out of scope** make sure angular out of the box unit testing is not throwing any errors.

```
ng test
```

Performance

High performance is the key thing that is expected out of any angular application. Below pointers helps you get the optimal performance.

- First Paint: Only the root `AppModule` should import the `CoreModule`.

- Lazy Loading : A distinct application feature or workflow may be *lazy loaded* or *loaded on demand* rather than when the application starts.
- Bundling/Minification
- Caching
- Pre-fetching of resources wherever necessary
- Routing

Tools/Plug-ins

- Plug-ins for VS Code

Using the following plug-ins will make the developer more productive. Names of the plugins are given here, you can get more details about what they do while you are installing them from marketplace/npm.

- **TS Lint** : Triggers TypeScript lint errors while you code and also provides Autofix capabilities in VS Code.
- **Auto Import**: Auto Import of ts files when you use it like modules, components, directives, services
- **Angular productivity Pack**: which installs the bunch of plugins including snippets, simontest, VS language extensions for typescript & javascript.
- **Bracket Pair Colorizer** : Nested brackets are colored appropriately which makes it easier to code.
- **Path Intellisense**: Enables path intellisense on HTML, CSS, JS, TS files when you need to import other files
- **VSCode-Icons**:Loads of icons with many tailor made for Angular features used (spec, service, component, module, model)
- **Debugger for Chrome**:This is must have to debug your code from chrome right from VS Code
- **Auto Code Formatting**: Developers should spend their productive time in building applications and leave code formatting, styling required by linters to automation tools. One such tool which works well for Angular, HTML5, CSS, Sass, Less.is prettier.
  Use the following prettier config for Angular

  ```
  {
   printWidth: 120,
   singleQuote: true,
   useTabs: false,
   tabWidth: 2,
   semi: true,
   bracketSpacing: true
  }
  ```

  For VS Code, install the following extension

  ```
  code --install-extension esbenp.prettier-vscode
  ```

  Finally enable auto formatting on save in VSCode through user settings

  ```
  editor.formatOnSave: true
  ```

- Tools

- **Chrome Dev Tools Audits**
  use **Chrome Dev Tools Audits** tab (Lighthouse Tool for Web Developer), it helps in improving the quality of the web page, by auditing for performance, best practices, accessibility & other aspects like PWA configuration. More details for the same are available on **here**.
- **Codelyzer**
  A set of tslint rules for static code analysis of Angular TypeScript projects. More details on the same are available on **here**.

Unused code

After integration or logical conclusion during the coding, make sure to remove all unused references,code blocks & files which makes the production bundle smaller and optimal.